remote sensing

MDPI

Article

GPU-Based Parallel Implementation of VLBI Correlator for Deep Space Exploration System

Fan Zhang [1,2], Chenxi Zhao [1], Songtao Han [3], Fei Ma [1,*] and Deliang Xiang [2]

[1] College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China; zhangf@mail.buct.edu.cn (F.Z.); buct_zcx2014@163.com (C.Z.)
[2] Interdisciplinary Research Center for Artificial Intelligence, Beijing University of Chemical Technology, Beijing 100029, China; xiangdeliang@mail.buct.edu.cn
[3] Beijing Aerospace Control Center, National Key Laboratory of Science and Technology on Aerospace Flight Dynamics, Beijing 100094, China; justdoit_doing@126.com
* Correspondence: mafei@mail.buct.edu.cn

Abstract: Very Long Baseline Interferometry (VLBI) solution can yield accurate information of angular position, and has been successfully used in the field of deep space exploration, such as astrophysics, imaging, detector positioning, and so on. The increase in VLBI data volume puts higher demands on efficient processing. Essentially, the main step of VLBI is the correlation processing, through which the angular position can be calculated. Since the VLBI correlation processing is both computation-intensive and data-intensive, the CPU cluster is usually employed in practical application to perform complex distributed computation. In this paper, we propose a parallel implementation of VLBI correlator based on graphics processing unit (GPU) to realize a more efficient and economical angular position calculation of deep space target. On the basis of massively GPU parallel computing, the coalesced access strategy and the parallel pipeline strategy are introduced to further accelerate the VLBI correlator. Experimental results show that the optimized GPU-based VLBI method can meet the real-time processing requirements of the received data stream. Compared with the sequential method, the proposed approach can reach a 224.1× calculation speedup, and a 36.8× application speedup. Compared with the multi-CPUs method, it can achieve 28.6× calculation speedup and 4.7× application speedup.

Keywords: very long baseline interferometry; graphics process unit; massively parallel; deep space exploration

check for updates

Citation: Zhang, F.; Zhao, C.; Han, S.; Ma, F.; Xiang, D. GPU-Based Parallel Implementation of VLBI Correlator for Deep Space Exploration System. *Remote Sens.* **2021**, *13*, 1226. https://doi.org/doi:10.3390/rs13061226

Academic Editors: David López Vilariño, Francisco Fernández Rivera, José Carlos Cabaleiro Domínguez and Tomás Fernández Pena

Received: 19 February 2021
Accepted: 19 March 2021
Published: 23 March 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The computational demands of scientific research are constantly increasing. In the field of radio astronomy, observation has evolved from the single telescope to the interferometer arrays, which is currently under development. At the same time, the Very Long Baseline Interferometry (VLBI) technology, which increases the accuracy of observation by extending the distance between the stations, has also been rapidly developed. They are all developed based on interferometric technology, but the data processing methods of them are different.

VLBI plays a key role in deep space exploration and astronomical observation due to the capability of high accurate angular measurement. The technology performs correlation calculations on the observation data of multiple radio telescopes, thus synthesizing multiple telescopes into a synthetic aperture telescope with an equivalent diameter of the longest baseline length [1,2]. Until now, the VLBI has been widely used in many fields, such as geodetic survey, astrophysics, detector positioning, and so on [3,4]. With the increase of VLBI stations and the observation bandwidth, there is a strong demand for fast correlation processing [5,6].

For the interferometric technology, the most important part is the design of correlators. In the past few decades, many correlators based on dedicated hardware have been devel-

oped. Among various dedicated hardware, application specific integrated circuit (ASIC), digital signal processing (DSP) and field-programmable gate arrays (FPGA) are usually employed in the development of VLBI correlators. A. R. Whitney et al. [7] implemented a VLBI system based on the the custom-VLSI chip and the DSP. Compared with ASIC, FPGA has a shorter design cycle, lower development cost and more flexible design. Rurik A. Primiani et al. [8] designed a new VLBI correlator, which replaced the previously-coined ASIC correlator. Hitoshi Kiuchi et al. [9] proposed six preset correlation processing system on the basis of FPGA. In order to avoid programming, testing and debugging issues in the traditional FPGA development process, Gan et al. [10] adopted the OpenCL tool flow to convert the OpenCL kernel function into customized FPGA hardware accelerator files, automatically. DSP is more suitable for processing computing tasks, and the FPGA is better for logic operations. Shanghai Astronomical Observatory (SHAO) developed the third-generation Chinese VLBI Data Acquisition System (CDAS), and then equipped the DSP board with Xilinx XC7K480T FPGA for data processing [11]. The joint research activity in the RadioNet FP7 Programme designed a generic high-performance computing platform based on DSP and FPGA for radio astronomy, named UniBoard [12]. Although the methods based on dedicated chips can effectively implement the VLBI correlator, their economic costs and scalability are still insufficient.

A recent trend is to correlate in software instead of dedicated hardware. With the rapid development of high-performance computing technology [13–16], the correlators have begun to be developed on the general CPU platform. In the earlier work, VLBI software correlators are implemented based on a single CPU [17], e.g., the VLBI correlator from National Radio Astronomy Observatory (NRAO) was developed based on an IBM 360/50 [18]. Furthermore, the VLBI correlators are implemented on CPU clusters to achieve higher efficiency, such as the Chinese VLBI Network (CVN) Earth software developed by Shanghai Astronomical Observatory of the Chinese Academy of Sciences [19] and DiFX VLBI correlator proposed by Swinburne University of Technology [20]. Extensive experiments verify that the CPU cluster can realize real-time VLBI data processing at a medium data rate, and it is easier to design and develop compared with the dedicated chips.

The emergence of the graphics processing unit (GPU) and the compute unified device architecture (CUDA) provide new opportunities for accelerating VLBI correlators [21,22]. GPU has been developed as a high-performance device with the characteristics of high parallelism, multithreading, many-cores, huge bandwidth capacity, and hundreds of computing cells. But they only discussed how to deploy the algorithm on heterogeneous platforms and analyze the processing results of the software. In recent years, for the interferometric array observation technology, some correlators have been accomplished on the GPU. Thomas Hobiger et al. [23] discussed the feasibility of implementing correlator on GPU. The Cobalt project implemented a GPU-based correlator for LOFAR [24]. However, the GPU-based parallel acceleration method of VLBI algorithm has not been studied yet. Besides, GPU has been widely used in the field of remote sensing for big data processing [25–28]. Inspired by the remote sensing applications and correlators of interferometric array observation technology on GPU, we try to implement a high efficient GPU-based VLBI correlator considering hierarchical optimization strategies.

In order to meet real-time processing at high data rate, we propose a parallel implementation of VLBI algorithm on the CPU-GPU heterogeneous platform. To solve the computation-intensive issue, the VLBI algorithm is implemented in GPU massively parallel mode, and is designed to realize the fine-grained task partitioning and calculation. Further, to deal with the data-intensive problem, the multi-level GPU optimization strategies have been introduced to improve the efficiency of data access and processing pipeline. Finally, the proposed method has been evaluated by the actual observation data. It is proved that the GPU implementation can meet the demands of high data rate real-time calculation, and is superior to the serial CPU method and multi-core CPU method.

In all, compared with previous works, we make the following contributions.

(1) GPU parallelization of the VLBI algorithm is presented. According to the characteristics of the GPU, multi-point calculation tasks are allocated to GPU threads to make full use of the computing capability of the threads and the hardware resources of the GPU.
(2) Shared memory is used to solve the uncoalescing problem encountered when accessing global memory.
(3) According to the characteristics of VLBI data, CUDA stream is used to optimize the data transmission between CPU and GPU.

The remainder of this paper is organized as follows. Section II presents the calculation flow of the VLBI algorithm and analyzes the algorithm complexity. Section III gives the details of the proposed GPU-based parallel VLBI method. Experimental results are shown and analyzed in Section IV. Conclusions and perspectives are given in Section V.

2. Methodology

In this section, the principle and implementation of VLBI correlator are introduced specifically. Meanwhile, the analysis of the algorithm complexity is also briefly analyzed.

2.1. The Principle of VLBI

The physical process of VLBI system is shown in Figure 1. The two ground stations simultaneously receive signals from the same observation target. Then, each station sends the observation data to the VLBI correlator, which performs correlation calculations to obtain the residual delay. Finally, the residual delay and estimated delay are employed to calculate the angular position of the observed target (i.e., θ). Through the above accurate coherent processing, sub-millisecond spatial resolution and nanosecond delay estimation can be obtained.

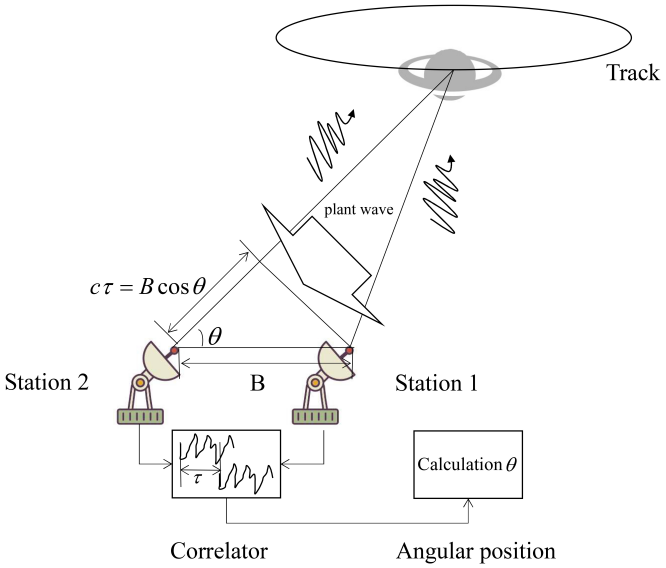

Figure 1. The illustration of Very Long Baseline Interferometry (VLBI) system.

2.1.1. The Physical Process of the VLBI

Because the distance between the observation target and the earth is long enough, the signals received by the station can be approximately regarded as the plane waves. In addition, due to the different geographic locations, the stations will receive the same wavefront signal at different time points. Therefore, there will be a delay between the two stations when receiving the same signal. According to the geometric relationship in Figure 1, the delay τ can be calculated as the following:

$$\tau = \frac{B\cos\theta}{c}, \tag{1}$$

where B is the baseline length between the two stations, c is the speed of light, and θ is the intersection angle between the direction of the observation target and the baseline, that is, the angular position information of the target. Accordingly, the orbital position of the target can be determined with θ. Therefore, τ needs to be calculated for obtaining the angular position of the observation target.

By calculating the derivative with respect to θ in Equation (1), the accuracy expression of the angular position θ can be obtained as below.

$$\partial\theta = \frac{c\partial\tau}{B\sin\theta}. \tag{2}$$

It can be seen from Equation (2) that the accuracy of the angular position θ is directly proportional to the delay accuracy $\partial\tau$ and inversely proportional to the baseline length B. Therefore, the higher angular position accuracy can be obtained by increasing the baseline length or the delay accuracy.

2.1.2. The Signal Model of VLBI Correlator

As aforementioned, there is a time delay between the signals of the two stations that receive the same target signal. To acquire a more accurate delay, the following two steps are performed. Firstly, some compensation methods are utilized to roughly compensate for the delay between the signals of the stations. Then, the residual delay can be obtained through the correlation operations of the compensated signal. Next, the single-frequency signal model is employed to introduce the principle of VLBI method.

First, the real delay between two stations is assumed to be τ_g. The signals received by the two stations from the observation target are expressed as follows:

$$x_1(t) = \cos\omega t, \tag{3}$$

$$x_2(t) = \cos\omega(t-\tau_g), \tag{4}$$

where ω is the carrier frequency of the observation signal, and t is the time scale. After the carrier synchronization, the signals of the two stations can be expressed as:

$$x_1(t) = \cos(\omega-\omega_0)t, \tag{5}$$

$$x_2(t) = \cos[(\omega-\omega_0)t-\omega\tau_g], \tag{6}$$

where ω_0 is the frequency of local oscillator. After the rough delay compensation of the signal, the signal of Station 2 can be obtained as follows:

$$x_2^{'}(t) = \cos[(\omega-\omega_0)(t+\tau_{g0})-\omega\tau_g], \tag{7}$$

where τ_{g0} is the rough delay estimation of the actual delay compensation model. The rough delay value is usually calculated by the empirical formula of fifth degree polynomial.

Then, based on the rough delay compensation, the result of correlation processing is as follows:

$$\begin{aligned} R &= x_1(t)\otimes x_2^{'}(t) \\ &= \cos[(\omega-\omega_0)\tau_{g0}-\omega\tau_g] \\ &= \cos[-(\omega_0-\omega)(\tau_g-\tau_{g0})-\omega_0\tau_g]^{'} \\ &= \cos[(\omega-\omega_0)\Delta\tau_g+\omega_0\tau_g] \end{aligned} \tag{8}$$

where $\Delta\tau_g$ is the residual delay value. It can be seen from the correlation results that the phase contains a linear component that varies with frequency. The slop of the phase-frequency diagram is just the residual delay, which can be employed to calculate the angular position.

Based on the residual delay calculated from the above processing, the angular position information of the observation target can be obtained by calculation with Equation (1).

$$\theta = \arccos \frac{c(\Delta\tau_g + \tau_{g0})}{B}. \tag{9}$$

2.2. The Implementation of VLBI Correlator

As shown in Figure 2, the VLBI algorithm mainly includes four parts, respectively, rough delay calculation, delay compensation, correlation, and residual delay calculation. Through the above calculations, the real-time angular position can be acquired for the deep space exploration system.

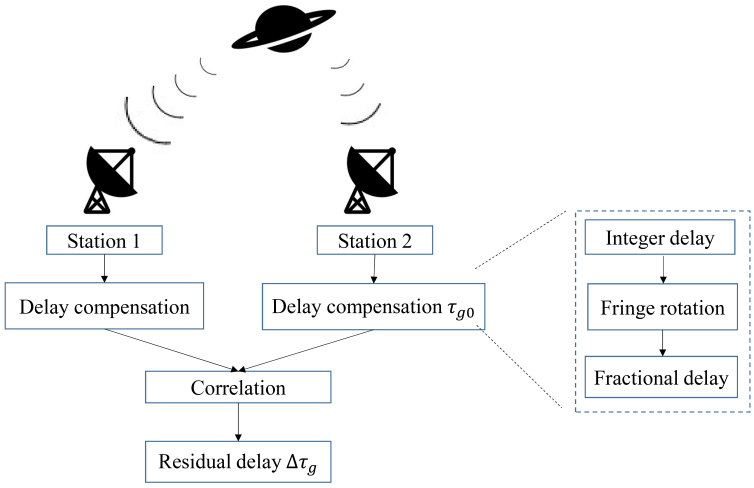

Figure 2. The structure of VLBI algorithm.

2.2.1. Rough Delay Calculation

The orbit prediction [29,30] method is used to estimate the delay compensation τ_{g0} between the two stations. Generally, the empirical formula of fifth degree polynomial is accurate enough to calculate the rough delay of any sampling time t,

$$\begin{aligned} \tau_{g0} &= a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5 \\ &= {\tau_{g0}}^I + {\tau_{g0}}^F \\ &= N_i * T_s + N_f * T_s \end{aligned}, \tag{10}$$

where a_i (i = 0, 1, 2, 3, 4, 5) is the empirical coefficients, ${\tau_{g0}}^I$ and ${\tau_{g0}}^F$, respectively, indicate the integer and fractional delay, N_i and N_f represent the number of sampling periods for the integer and fractional delay, and T_s is the sampling period.

2.2.2. Delay Compensation

Because the delay of the two stations is not always an integer multiple of the signal sampling period, the delay compensation can be divided into integer delay and fractional delay. Normally, the integer delay is compensated with the time domain shift, and the fractional delay is compensated through the frequency domain phase multiplication. Meanwhile, it is crucial to eliminate the influence of the Doppler effect in the delay compensation process, which corresponds to the fringe rotation part in the VLBI algorithm. As shown in Figure 2, the delay compensation includes three steps, including the integer delay compensation, the fringe rotation and the fractional delay compensation. The specific implementations are as follows.

(i) Integer delay compensation
The signals of one station cope with the integer delay as follows:

$$\boldsymbol{I}(n) = \boldsymbol{x}(n + N_i), \tag{11}$$

where n indicates the discrete time scale, $\boldsymbol{x}(n)$ is the received signal, and $\boldsymbol{I}(n)$ indicates the processed signal after the integer delay compensation.

(ii) Fringe rotation
The processing step is introduced to counteract the influence of the Doppler effect, which is denoted as follows:

$$\boldsymbol{Fri}(n) = \boldsymbol{I}(n) \times e^{j2\pi \cdot f_d(n)}, \tag{12}$$

where $\boldsymbol{Fri}(n)$ is the result after eliminating the Doppler effect, and $f_d(n)$ is the Doppler frequency employed in the estimation.

(iii) Fractional delay compensation
After carrying out the fractional delay compensation, the signal can presented below:

$$\boldsymbol{Fra}(f) = FFT(\boldsymbol{Fri}(n)) \times e^{-j2\pi \cdot f \cdot \tau_{g0}^{F}}, \tag{13}$$

where f indicates the discrete frequency scale, and $FFT(\cdot)$ means the Fourier transform operator.

2.2.3. Signal Correlation

Through the above operations, the compensation operations of the signals are accomplished. According to Equation (8), the correlation between the two stations can be executed as below:

$$\boldsymbol{P}(f) = \boldsymbol{Fra}(s1)(f) \times \bar{\boldsymbol{Fra}}(s2)(f) = |P(f)|e^{j\varphi(f)}, \tag{14}$$

where $s1$ and $s2$ indicate the different stations that observing the same target.

2.2.4. Angular Position Calculation

After finishing the above operations, the residual delay can be obtained through calculating the slop of the phase spectrum in the correlation results.

$$\Delta\tau_g = \frac{\partial\varphi(f)}{2\pi \cdot \partial f}. \tag{15}$$

After solving the two values τ_{g0} and $\Delta\tau_g$, the angular position can be figured out by Equation (9).

2.3. The Analysis of Algorithm Complexity

According to the previous VLBI introduction, the complexity analysis of the four steps will be given successively [31]. As for the calculations of rough delay, it is independent of signal length L, and its computational complexity is $O(1)$. Next, the integer delay compensation can be implemented through the data shift operation, which can be implemented along with the data reading process. Therefore, the computational complexity of this part can be seen as $O(1)$. As Equation (12) described, the essence of fringe rotation includes the Doppler frequency calculation of L points and the multiplication of L points in time domain. In the next step, the Fourier transform of L points and the multiplication will be employed to finish the fractional delay compensation. Accordingly, the computational complexity can be expressed as $O(LlogL)$. The algorithm complexities of the correlation calculation and the slop estimation are related to the signal length L.

As aforementioned, the calculation of VLBI mainly includes the multiplication and Fourier transform in the signal dimension. For the consideration of efficiency, Intel

Integrated Performance Primitives (IPP) [32,33] is employed to implement these time-consuming steps in a serial way. Figure 3 depicts the time-consuming ratio of the serial VLBI algorithm. It can be seen that the time-consuming situation of the fringe rotation and fractional delay are slightly different from the above complexity analysis. The main reasons include two aspects: first, due to the application of Intel IPP library library, the operations of FFT and sequence multiplication are optimized at the software and hardware level; second, although the complexity is the same, the amount of calculation is different between the fringe rotation and the fractional delay. Thus, the fringe rotation takes more time in the implementation. In general, the most time-consuming part is concentrated on the calculation of delay compensation and correlation, which accounting for about 99.98% of the total running time.

In practical applications, in order to improve the signal-to-noise ratio and acquire a more accurate delay, the signal is divided into multiple blocks. Concretely, the signal-to-noise ratio will be enhanced by accumulating the results of these blocks. First of all, the pending signals of one station which contains C channels are divided into blocks as follows. There are Z signals in one channel. Each signal is divided into A integration time; an integration time includes F FFT time; and one FFT time has N discrete points. According to the above definitions of the blocks, there are M (i.e., $M = Z \times A$) integration time in one channel. Algorithm 1 shows the pseudo-code of the serial VLBI algorithm, which processes the data of one channel. From Algorithm 1, the main time consumption comes from the two four-layer loops. Combine the analysis of time complexity and Algorithm 1, our experiment mainly optimizes the operations of delay compensation and correlation in parallel.

Algorithm 1: Serial version of VLBI.

Input: The data of S stations $\boldsymbol{x}[n]$
Output: The residual delay

Step1:
for *each* $s \in [1, S]$ **do**
  $\boldsymbol{I}[s][n] = \boldsymbol{x}[s][n + N]$
end
Step2:
for *each* $a \in [1, M]$ **do**
  for *each* $s \in [1, S]$ **do**
    for *each* $i \in [1, F]$ **do**
      for *each* $n \in [1, N]$ **do**
        $\boldsymbol{Fri}[a][s][i][n] = \boldsymbol{I}[a][s][i][n] \times e^{j2\pi f_d[n]}$
      end
      for *each* $n \in [1, N]$ **do**
        $\boldsymbol{Fri}^{\boldsymbol{f}}[a][s][i] = FFT(\boldsymbol{Fri}[a][s][i])$
        $\boldsymbol{Fra}[a][s][i][n] = \boldsymbol{Fri}^{\boldsymbol{f}}[a][s][i][n] \times e^{-j2\pi\tau_{g0}^{F}[n]}$
      end
    end
  end
end
Step3:
for *each* $z \in [1, Z]$ **do**
  $\boldsymbol{P}[z] = \boldsymbol{Fra}[s1][z] \times \bar{\boldsymbol{Fra}}[s2][z]$ ← *Calculate the Correlation between two stations (s1 and s2)*
end
Step4:
for *each* $z \in [1, Z]$ **do**
  Calculate the residual delay
end

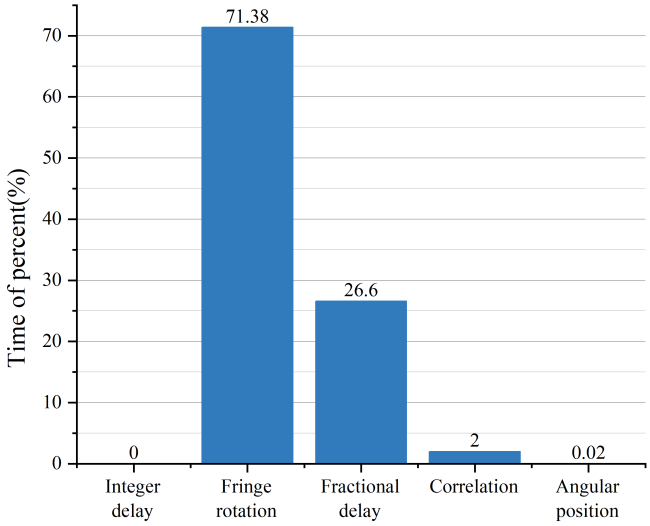

Figure 3. Time proportion of each part in the serial version.

3. Proposed Method

It can be seen from the above analysis that the fringe rotation, fractional delay compensation, and signal correlation are the most time-consuming parts of the VLBI algorithm. In order to accelerate the VLBI algorithm, the three parts are parallelized in the GPU. The GPU parallel version of VLBI is displayed in Figures 4 and 5. According to the characteristics of the GPU, it has thousands of physical threads, so it is more suitable for performing data-intensive and computationally intensive operations. It can be seen from Figure 4 that the CPU-based parallel VLBI algorithm may require multiple devices to achieve the processing effect of the GPU-based parallel algorithm.

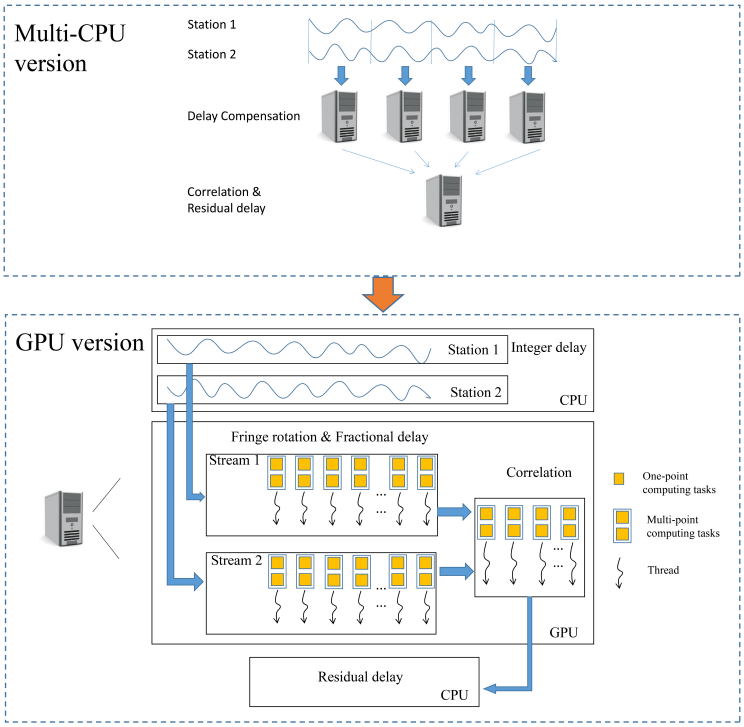

Figure 4. Illustration of parallel VLBI based on multilevel graphics processing unit (GPU) optimization.

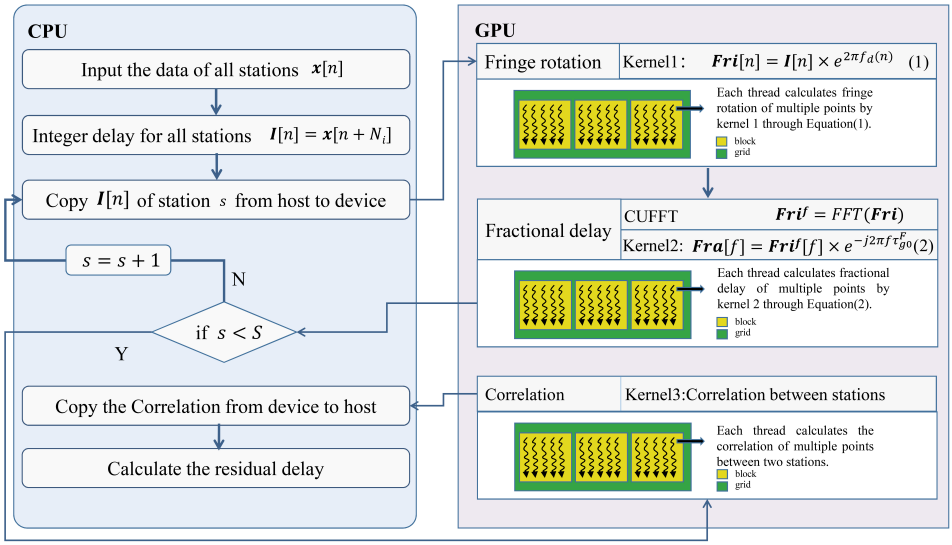

Figure 5. GPU-based parallel VLBI framework.

As shown in Figure 5, it can be seen that the calculations of rough delay, the integer delay, and the angular position are executed on the CPU. Besides, the operations of fringe rotation, fractional delay and correlation are executed on the GPU. After the calculation of correlation is completed, the results will be returned from the GPU to the CPU. Eventually, the phase and frequency in the correlation results are fitted to a straight line to obtain the residual delay on the CPU.

3.1. The Parallel Optimization Strategies

3.1.1. The CPU-GPU Data Transfer Optimization

According to the CUDA programming paradigm, the data transfer is necessary between CPU and GPU. Basically, the required input data is transferred from CPU to GPU for the parallel function. After the high performance computing, the results will be transferred from the GPU to the CPU. Since the peak bandwidth between the CPU memory and the GPU memory is relatively low, it means that the data transfers between the CPU and GPU may slow the whole application performance. The following are some general guidelines for the data transfer between them.

- Minimize the number of transfers between the host and the device.
- Merge many small transfers into one.

In this paper, we will treat the data in an integration time as a basic data unit, and transfer it to the GPU, which will eliminate most of the per-transfer overhead.

3.1.2. The Solution of the Uncoalescing Problem

Since the data type is complex, when we operate on the real or imaginary part of the data on the global memory, the uncoalescing problem will be encountered. As shown in Figure 6, the discontinuous storage areas of global memory are accessed, and will lead to the performance decline. If the data access is continuous, one instruction can read more data. This means fewer instructions will be used to perform data access operations. Thus, the data reading time will be shortened. Memory coalescing technology can greatly improve the speed of global memory access. The hardware will detect whether the location accessed by the thread is an adjacent location in the global memory. When 16 threads of a warp are coalesced in memory transactions, the bandwidth of the global memory will be maximized. Therefore, shared memory is employed to solve the uncoalescing problem in this article, as shown in Figure 6. We employ a continuous method to access data from global memory to shared memory, firstly. Since there is no need to consider the order of access when using shared memory, the problem of uncoalescing problem can be avoided.

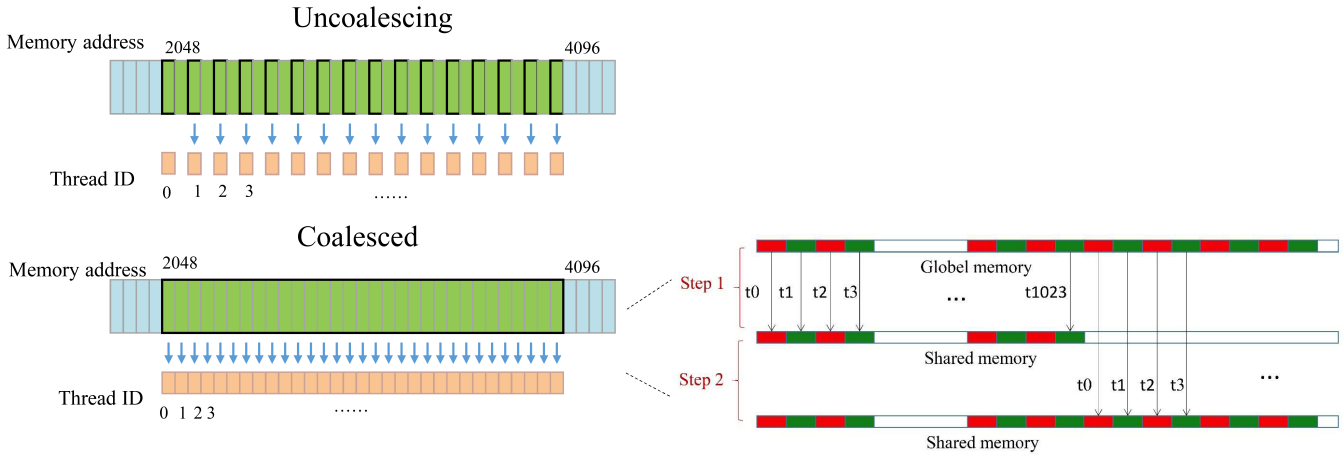

Figure 6. The uncoalescing problem and the corresponding solution.

When a block starts to execute, the GPU will allocate a certain amount of shared memory, and the address space of the shared memory will be shared by all threads in the block. Shared memory is allocated to all blocks in Streaming Multiprocessing (SM), which is also a scarce resource of the GPU. Therefore, the more shared memory is employed, the less active threads that can be parallelized.

The main operations of memory coalescing method is shown below. Firstly, adjacent threads read the neighboring real or imaginary number from global memory to shared memory. Then, in the kernel function, the data in the shared memory is converted into a complex type and calculated. Finally, the complex number is divided into the real and imaginary part for output.

3.1.3. The Thread Task Allocation

The main content of a parallel algorithm is the task partitioning and scheduling, namely, the task allocation on each thread. When formulating tasks to be executed on each thread, the following two aspects are mainly considered. First, the GPU thread has limited computing ability. Nevertheless, when designing the thread computing tasks, it must give full play to its capabilities within the scope of its computing power. Secondly, if more resources (such as shared memory, register, etc.) are used in a thread block, fewer active thread blocks that can be parallelized.

Based on the parallel design guidelines, the calculation of multiple data points will be considered to execute in one thread. To optimize the performance of the GPU implementation, the maximum number of threads, namely 1024, is set to each thread block. Accordingly, the block number can be calculated as $(F \times N/(1024 \times N_t))$, where F and N_t are the dimension of Fourier transform and the number of points computed in one thread, respectively.

3.1.4. The Cuda Stream

It is noted that the data calculation process of each station is independent in the serial algorithm. In order to optimize the execution pipeline, the CUDA stream technology is introduced to realize concurrent execution of parallel calculation and data transmission. Thus, the overall running time can be decreased greatly. From Figure 7, it can be seen that CUDA can start multiple streams, simultaneously. The data transmission and kernel calculation are performed in each stream, independently. In this way, when the data of current station performs the calculation, the data of next station can transmit data or execute the calculation operation. In the GPU parallel implementation, the parallel pipeline strategy outperforms the serial pipeline by hiding the data transfer overhead.

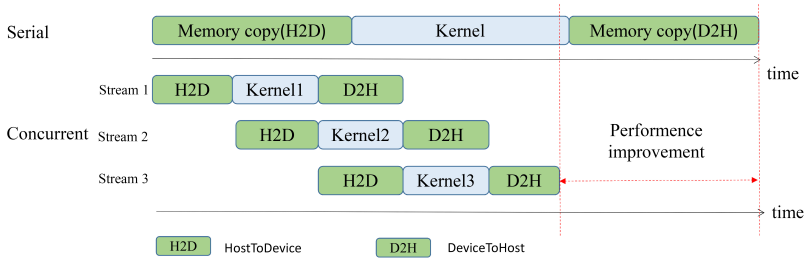

Figure 7. The principle of compute unified device architecture (CUDA) stream.

3.2. The Overall Approach of Gpu Parallel VLBI

A detailed step-by-step algorithm description of the parallel VLBI on the CPU–GPU heterogeneous platform is summarized in Algorithm 2. The calculations of fringe rotation, fractional delay, and correlation are denoted as kernel 1, kernel 2 and, kernel 3, which are executed on the GPU. The CUFFT library is used to perform FFT operations on the GPU. Besides, the CUDA stream is used to reduce the time of the data transmission.

Algorithm 2: Parallel version of VLBI.

Input: The data of S stations $\boldsymbol{x}[n]$
Output: The residual delay

Step1:
for *each* $s \in [1, S]$ **do**
  $\boldsymbol{I}[s][n] = \boldsymbol{x}[s][n + N]$
end
Step2:
for *each* $a \in [1, M]$ **do**
  Start CUDA stream
  for *each* $s \in [1, S]$ **do**
    kernel 1:
    Copy $I[a][i]$ ***from the CPU to the GPU***
    The following part runs on GPU
    apply for 1D CUDA block and thread
    $\boldsymbol{Fri}[a][s][threadId] = \boldsymbol{I}[a][i][threadId] \times e^{j2\pi f_d t}$
    end kernel 1
    $\boldsymbol{Fri}^f[a][s] = CUFFT(\boldsymbol{Fri}[a][s])$
    kernel 2:
    apply for 1D CUDA block and thread
    $\boldsymbol{Fra}[a][s][threadId] = \boldsymbol{Fri}^f[a][i][threadId] \times e^{-j2\pi f \tau_{gF}}$
    end kernel 2
  end
  end CUDA stream
end
Step3:
for *each* $z \in [1, Z]$ **do**
  kernel 3:
  $\boldsymbol{P}[z] = \boldsymbol{Fra}[z][s1] \times \bar{\boldsymbol{Fra}}[a][s2] \leftarrow$ *Calculate the Correlation between two stations (s1 and s2)*
  end kernel 3
end
Step4:
for *each* $z \in [1, Z]$ **do**
  Calculate the residual delay
end

4. Results

4.1. Experimental Configuration

The hardware specification is described in Table 1. It also shows the bandwidth of data transfer between the CPU and the GPU. The software environment includes Windows 10 64 bit operating system, Visual Studio 2017, CUDA 9.2, and Intel IPP library. The experimental data are from Beijing Aerospace Control Center. The observation stations are Kashi Station and Jiamusi Station, respectively. The data of each station has 8 channels, and the data rate is 64 Mb/s. The sampling rate and quantization bit are 4 MHz and 1 bit. The observation time of the experimental data is 6 s.

Table 1. Experimental platform hardware specifications.

CPU	CPU version	Intel i5-4950
	Processor base frequency	3.3 GHz
	CPU memory	DDR3
	Number of cores	4 in total (1 CPU)
	Main memory	8 GB
GPU	GeForce	NVIDIA RTX 2070
	Architecture	Turing
	Frequency of CUDA cores	1725 MHz
	Number of CUDA cores	2304
	Single precision floating point performance(peak)	7.5T Floats/s
	Dedicated memory	8 GB
	Memory speed	7001 MHz
	Memory interface	256-bit
	Memory bandwidth	448 GB/s
	CUDA compute capability	7.5
	Threads/warp	32
	Threads/block	1024
	Shared memory/block	49,152 bytes
	Registers/block	65,536
Bandwidth	Host To Device	11,891.6 MB/s
	Device To Host	10,054.8 MB/s
	Device To Device	374,734.7 MB/s

4.2. Efficiency Analysis

In the experiments, the proposed GPU-based VLBI algorithm (denoted as GPU version) are compared with the corresponding serial version and the multi-CPU version. The serial version is implemented in C language based on the intel IPP library. The multi-CPU version of VLBI (denoted as multi-CPU version) is implemented with the message passing interface (MPI) and the Intel IPP library. We evaluate the performance improvement (acceleration factors) of the GPU implementation relative to the multi-CPU version and the serial version.

4.2.1. The Analysis of the Optimization Strategies

Figure 8 depicts the calculation time of fractional delay and fringe rotation for one channel data of a station. Through shared memory bridging, the problem of discontinuous data access is solved. When reading data from global memory, the bandwidth of the global memory is fully utilized. After solving the uncoalescing problem, the execution time of the algorithm is significantly reduced. From Figure 8, it can be seen that the processing time of fringe rotation and fractional delay is reduced along with the decrease of the block number. If multi-point computing tasks are allocated on a thread, the computing resources of the thread can be fully utilized, and sufficient hardware resources can be allocated for each thread. The aforementioned optimization methods can reduce the time of processing data. Furthermore, the results verify the effectiveness of the method proposed. For the fractional delay, the time change is more obvious after optimization. According to the analysis of the algorithm in Section II, although the complexity is the same, the amount of calculation is different between the fringe rotation and the fractional delay. Therefore, the thread in Kernel1 has a larger amount of calculation and consumes more hardware resources. Thus, the repartitioning of GPU resources has less impact on edge rotation.

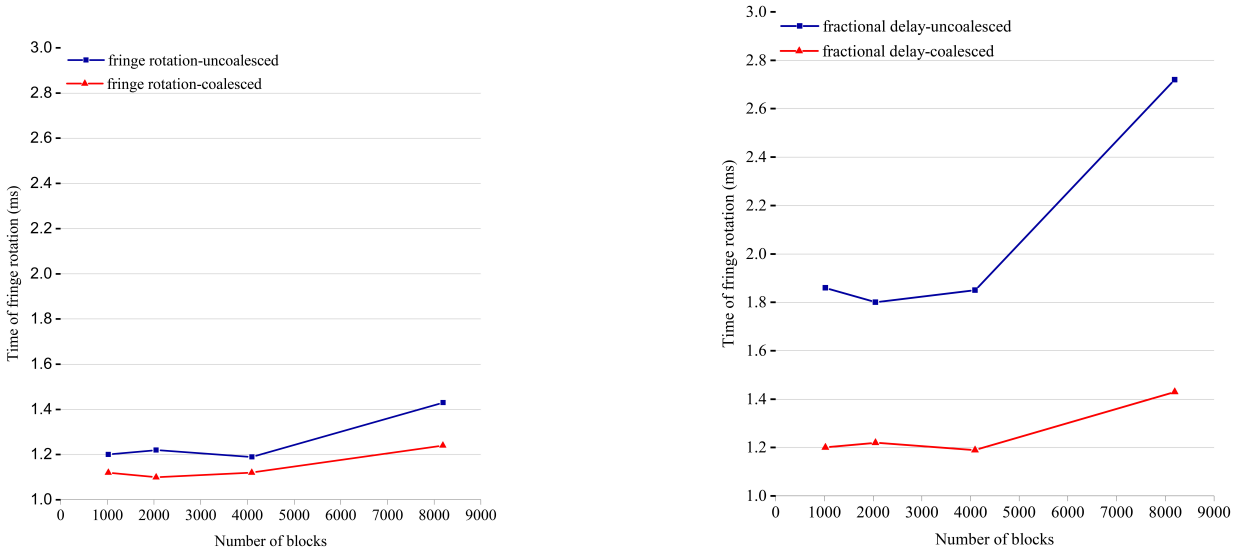

Figure 8. The thread allocation optimization under different number of blocks. (**a**) Fractional delay. (**b**) Fringe rotation.

Figure 9 shows the percentages of the data transmission time and the calculation time. It can be observed that the data transfer time is reduced after applying the proposed parallel pipeline. In particular, after using CUDA stream for transmission optimization, the time percentage of data transfer has been reduced from 90.37% to 66.86%. However, the time for data transfer still accounts for a large part of the total time. That is because the CUDA stream operation only hides a part of the data transfer time between the CPU and GPU, but it does not shorten it, as shown in Figure 7. In addition, when optimizing without using CUDA streams, the data transmission time accounts for more than 90% of the total time. In view of the above two reasons, the proportion of data transmission time still accounts for a large proportion. Regardless, for the VLBI algorithm, the CUDA stream is capable of shortening the ratio of data transmission, and is essential to promote algorithm performance.

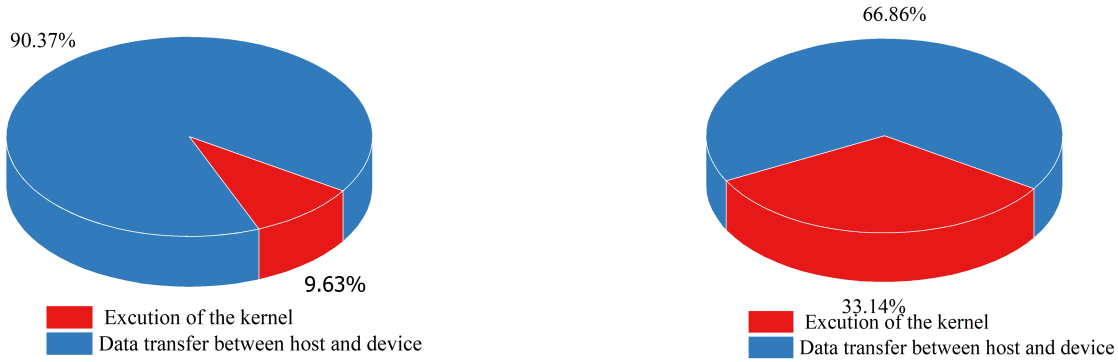

Figure 9. Proportion of kernel calculation and data transmission time. (**a**) Without the CUDA stream for optimization. (**b**) Optimization after using CUDA stream.

Table 2 indicates the time consumption and speedups of various optimization strategies applied in this article. The different approaches are employed to process the data in one channel. "Serial" refers to the serial algorithm running on the single CPU. Moreover, the model and frequency of the CPU are revealed in Table 1. In the version of "Initialization", the task performed on each thread is a single point calculation, where CUFFT is used to optimize the FFT. Based on the "Initialization" , "Coalesce" solves the uncoalescing problem when accessing the global memory. "Allocation" version redesigned the computing tasks performed on each thread in the GPU. The version of "CUDA stream" further optimize the data transmission between the CPU and the GPU.

According to the above results, the multilevel GPU optimization strategy is consistent with the design ideas. Based on the basic GPU parallel, the serial algorithm is accelerated by 18.1×. After solving the uncoalescing problem, the bandwidth of global memory is fully utilized. Hence, the "Coalesce" version attains the speedup of 19.1×. When making full use of thread computing resources and GPU hardware resources, the acceleration effect reaches 22.1×. Benefiting from the CUDA stream, multiple streams can be started at the same time to process data from different stations. Finally, compared with the serial version, the method proposed in this article can achieve 36.6×. It can be seen from the optimization results of the above acceleration method that data transmission has the most significant impact on the computational efficiency of the VLBI algorithm. Therefore, after using stream optimization, the acceleration effect is significantly improved.

Table 2. Speedup after using different optimization strategies (Result: The result of the residual delay of one channel data.)

Method	Time(ms)	Speedup	Result (s)
Serial	2461.5	1×	-1.28×10^{-11}
Initialization	135.6	18.1×	-1.28×10^{-11}
Coalesce	128.6	19.1×	-1.28×10^{-11}
Allocation	111.6	22.1×	-1.28×10^{-11}
CUDA stream	67.3	36.6×	-1.28×10^{-11}

4.2.2. The Comparison with Serial Version and Multi-CPU Version

Table 3 lists the specific time-consumption comparison of the serial version and the GPU version. The last two rows show that the time and speedups of "Calculation" (that is, only the calculation part, excluding data transmission time) and "Overall I" (including data transmission time), respectively. The part of fringe rotation achieves a high speedup of about 853.5×. And the fractional delay reaches about 98× speedup. The GPU parallel

implementation of the correlation achieves a speedup of 22.7×. Besides, the speedups of the "Calculation" and "Overall I" are about 224.1× and 36.8×, separately. As can be seen from Table 1, the GPU used in this paper (NVIDIA RTX 2070) has 2304 CUDA cores. In theory, it should be 2304× the execution speed of the serial version. However, the computing power of CPU and GPU threads si different. Besides, the data transfer between CPU and GPU is inevitable. Therefore, compared with the serial version, the GPU-based parallel version cannot achieve an acceleration of 2304×.

It is obvious from Table 3 that the GPU version achieves remarkable acceleration factors on the platform as compared to the serial version. As for the practical data processing, the execution time of the serial version is about 39s, which is far from meeting the requirements of real-time processing. However, the GPU-based method can fulfill the data processing in about 1s in that it takes full advantages of the computational power of GPUs, the high bandwidth, as well as low latency of shared memory, and benefits from exploiting the massively parallel nature of GPUs.

Table 3. The comparison between GPU version and serial version.

Component	Serial Version (ms)	GPU Version (ms)	Speedup
Integer delay	0	0	1×
Fringe rotation	28,164	33	853.5×
Fractional delay	10,496	107.2	98×
Correlation	790	34.8	22.7×
Residual delay	1.2	1	1×
Calculation	39,451.2	176	224.1×
Overall	39,451.2	1072	36.8×

To further evaluate the efficiency of the proposed parallel approach, we compare the GPU implementation with the multi-CPU correlator developed by Beijing Aerospace Control Center (denoted as the multi-CPU version in this article). The multi-CPU version runs on a workstation with two CPUs. The processor employed in the experiment is Intel(R) Xeon(R) CPU E5-2683 v3 with 28 threads (2 GHz). The workstation has 2 correlation computing nodes. Each node starts 20 threads to perform the computing tasks.

Table 4 is consistent with the content in Table 3. Compared with the fringe rotation, fractional delay and correlation of the multi-CPU version, the acceleration ratio of the GPU version has reached 124.7×, 6.4×, and 7×, respectively. Specifically, the part of "Calculation" and "Overall I" achieves about 28.6× and 4.7× speedup. It can be seen from the results that the multi-CPU version requires about 5 s to process data of 6 s observation time. Although the multi-CPU version can achieve real-time processing, the GPU version only needs 1s, which can further improve the processing capability. This means that it may take 5 such machines to achieve the GPU computing effect, as shown in Figure 4. The details of the time consumption on the three versions are displayed in Figure 10. It can be seen, in Tables 3 and 4, that the acceleration effect of the fringe rotation is more obvious than the fractional delay. The main reasons include two sides: firstly, the fractional delay contains the FFT operation, and the FFT operation has the highest time complexity; secondly, Compared with the Intel IPP library, the optimization effect of CUDA CUFFT has the limited optimization capabilities. Therefore, the optimization effect of the fractional delay is not as good as the fringe rotation.

Table 4. The comparison between GPU version and message passing interface (MPI) version.

Component	Multi-CPU Version (ms)	GPU Version (ms)	Speedup
Integer delay	0	0	1×
Fringe rotation	4114	33	124.7×
Fractional delay	686.7	107.2	6.4×
Correlation	243.2	34.8	7×
Residual delay	1.4	1	1.4×
Calculation	5045.3	176	28.6×
Overall	5045.3	1072	4.7×

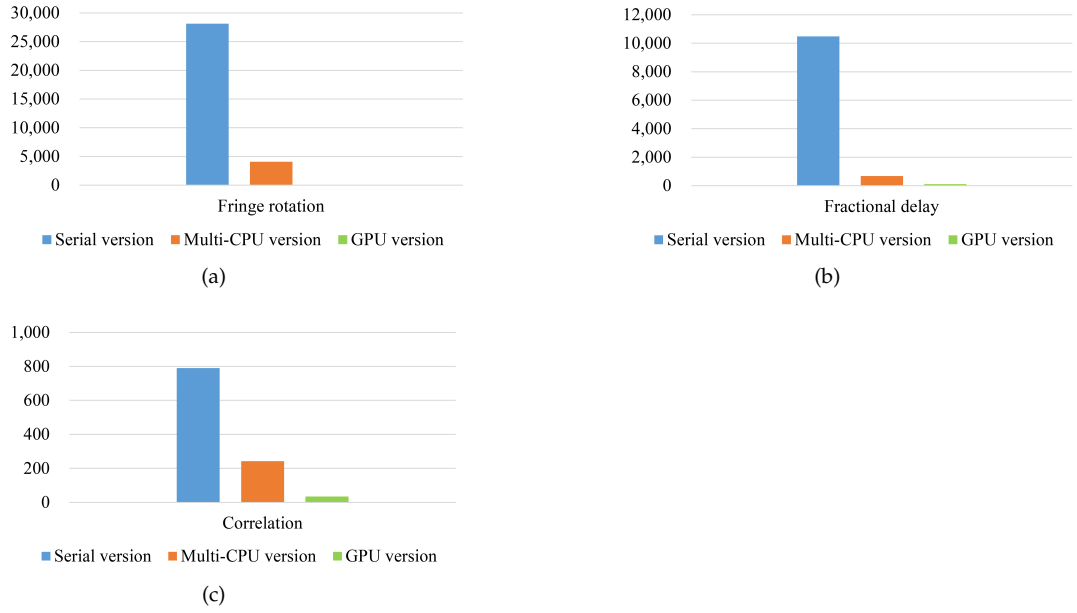

Figure 10. Execution time (ms) of each part in VLBI. (**a**) Fringe rotation. (**b**) Fractional delay. (**c**) Correlation.

4.2.3. The Comparison of Data Processing Rate

In the practical application, the data processing rate needs to reach 128 Mbps to meet the requirements of the real-time processing. From Table 5, it can be seen that the multi-CPU version has achieved the requirements of real-time processing. With the increase of data volume and data transmission rate, it will be difficult to undertake the future data processing. As for the GPU-based method, it has enough computing power margin to deal with this situation. Combining Figure 4 and Table 5, it can be summarized that the GPU-based VLBI algorithm can complete signal processing with fewer hardware resources and higher data processing efficiency than the multi-CPU version. Therefore, the GPU-based VLBI correlator will play a greater role in deep space exploration system.

Table 5. The comparison between GPU version and serial version.

Component	Running Time (ms)	Data Processing Rate (Mbps)
Serial version	39,451.2	19.3
Multi-CPU version	5045.3	151.2
GPU version	1072	711.7

4.3. Accuracy Analysis

VLBI technology aims to achieve higher observation accuracy by increasing the distance among stations. Therefore, the parallel algorithm design of the VLBI algorithm

should not only pay attention to the final acceleration effect, but also control the calculation error within an acceptable range.

Theoretically, there should be no difference between the results of the GPU-based method and the CPU-based method. In fact, the mathematical function libraries of the GPU and the CPU have different calculation accuracy. Furthermore, the instruction set optimization degree of CPUs and GPUs is different. Thus, their calculation results will exist a little difference in value.

As shown in Table 6, the value differences between CPU version and GPU version are listed for comparison. Mean absolute error (MAE) are employed to measure the difference of the two methods. It can be seen that the errors of FFT and fractional delay between two versions are both around 10^{-6}. The MAE difference of the residual delay step is about 5.1×10^{-15}. It can be observed from the overall MAE results that the calculation accuracy of the angular position can bee guaranteed.

Table 6. Differences in calculation results between serial version and parallel version.

Algorithm	MAE
Fringe rotation	0
FFT	0.8×10^{-6}
Fractional delay	1.75×10^{-6}
total	5.1×10^{-15}

5. Conclusions

In this paper, a GPU-based VLBI parallel correlator is proposed to meet the growing demands for high computing power. Through the algorithm complexity analysis, the targets of the parallel acceleration are determined, namely calculations of the fringe rotation, fractional delay and the correlation. By assigning the multi-point calculation as single thread task, the VLBI algorithm is massively parallelized. As a kind of computing-intensive and data-intensive issue, the GPU-based VLBI algorithm is further optimized in terms of data transfer. Through multi-level optimization strategies, namely data transmission rules, coalescing access and concurrent pipelines, the efficiency is increased by 2-fold. Under the premise of ensuring the calculation accuracy, compared with the serial version, the calculation part and the overall calculation have reached the speedups of 224.1× and 36.8×, respectively. Compared with the multi-CPU version, the corresponding parts have achieved speedups of 28.6× and 4.7×. In addition, the data processing rate of the GPU-based correlator can reach 712 Mbps, which can far meet the current ground station processing needs. The GPU-based correlator outperforms the actual multi-CPU-based correlator in terms of computing performance and data rate. In the future work, we will study the multi-GPU-based VLBI algorithm and deploy it to the better performance CPU cluster to further improve the data processing capability for deep space exploration.

Author Contributions: Conceptualization, F.Z. and S.H.; Formal analysis, F.Z. and C.Z.; Methodology, C.Z. and S.H.; Project administration, F.Z.; Supervision, F.Z., F.M. and S.H.; Visualization, C.Z. and D.X.; Writing—original draft, C.Z.; Writing— review and editing, F.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant 61871413, 41801236, and in part by the Fundamental Research Funds for the Central Universities under Grant XK2020-03.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the three anonymous reviewers for improving the quality of this article with their comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hewett, P. An introduction to radio astronomy. *Endeavour* **1997**, *21*, 134. [CrossRef]
2. Hao, W.; Dong, G.; Li, H.; Huang, L.; Zhou, H. The spacecraft signal correlation approach in China's Delta-DOR correlator for Chang'E-3 mission. In Proceedings of the 2014 IEEE Aerospace Conference, Big Sky, MT, USA, 1–8 March 2014; IEEE: New York, NY, USA, 2014; pp. 1–9.
3. Antreasian, P.G.; Baird, D.T.; Border, J.S.; Burckhart, P.D.; Graat, E.J.; Jah, M.K.; Mase, R.A.; Mcelrath, T.P.; Portock, B.M. 2001 Mars Odyssey Orbit Determination During Interplanetary Cruise. *J. Spacecr. Rocket.* **2005**, *42*, 394–405. [CrossRef]
4. Kroger, P.M.; Folkner, W.M.; Iijima, B.A.; Hildebrand, C.E. Very-long-baseline-interferometry measurements of planetary orbiters at Mars and Venus. *NASA STI/Recon Tech. Rep. A* **1993**, *95*, 1023–1037.
5. Hilliard, L.M.; Petrov, L.; LeMoine, F.; Rajagopalan, G.; Elosegui, P.; Ruszczyk, C.; Gipson, J.; Horsley, D.; Brown, G. 5 Year Technology Roadmap for VLBI Global Observing System (VGOS). In Proceedings of the IGARSS 2019—2019 IEEE International Geoscience and Remote Sensing Symposium, Yokohama, Japan, 28 July–2 August 2019; IEEE: New York, NY, USA, 2019; pp. 8956–8959.
6. García-Carreño, P.; García-Álvaro, S.; López-Pérez, J.; Patino-Esteban, M.; Serna, J.M.; Vaquero-Jiménez, B.; López-Fernández, J.; López-Espí, P.; Sánchez-Montero, R. Geodetic VLBI ultra low noise broad-band receiver for 13 meter VGOS radiotelescopes. In Proceedings of the 2016 11th European Microwave Integrated Circuits Conference (EuMIC), London, UK, 3–4 October 2016; pp. 476–479.
7. Whitney, A.R.; Cappallo, R.; Aldrich, W.; Anderson, B.; Bos, A.; Casse, J.; Goodman, J.; Parsley, S.; Pogrebenko, S.; Schilizzi, R.; et al. Mark 4 VLBI correlator: Architecture and algorithms. *Radio Sci.* **2004**, *39*, 1–24. [CrossRef]
8. Primiani, R.A.; Young, K.H.; Young, A.; Patel, N.; Wilson, R.W.; Vertatschitsch, L.; Chitwood, B.B.; Srinivasan, R.; MacMahon, D.; Weintroub, J. SWARM: A 32 GHz Correlator and VLBI Beamformer for the Submillimeter Array. *J. Astron. Instrum.* **2016**, *05*, 1641006. [CrossRef]
9. Kiuchi, H.; Imae, M.; Kondo, T.; Sekido, M.; Hama, S.; Hoshino, T.; Uose, H.; Yamamoto, T. Real-time VLBI system using ATM network. *IEEE Trans. Geosci. Remote Sens.* **2000**, *38*, 1290–1297. [CrossRef]
10. Gan, J.; Xu, Z. The Research of FPGA Acceleration for VLBI Hardware Correlator. In Proceedings of the 2018 Progress in Electromagnetics Research Symposium (PIERS-Toyama), Toyama, Japan, 1–5 August 2018; pp. 2088–2091.
11. Zhu, R.; Wu, Y.; Li, J.; Guo, S.; Gan, J.; Xu, Z. The Development of VLBI Digital Backend in SHAO. In Proceedings of the 2018 Progress in Electromagnetics Research Symposium (PIERS-Toyama), Toyama, Japan, 1–5 August 2018; pp. 1310–1314.
12. Hargreaves, J.E. UniBoard: Generic hardware for radio astronomy signal processing. In *Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy VI*; Holland, W.S., Ed.; International Society for Optics and Photonics, SPIE: Amsterdam, The Netherlands, 2012; Volume 8452, pp. 805–810. [CrossRef]
13. Liu, J.; Kang, X.; Dong, C.; Zhang, F. Simulation of Real-Time Path Planning for Large-Scale Transportation Network Using Parallel Computation. *Intell. Autom. Soft Comput.* **2019**, *25*, 65–77. [CrossRef]
14. M.Jamel, A.; Akay, B. A Survey and Systematic Categorization of Parallel K-means and Fuzzy-c-Means Algorithms. *Comput. Syst. Sci. Eng.* **2019**, *34*, 24.
15. M.Jamel, A.; Akay, B. Human Activity Recognition Based on Parallel Approximation Kernel K-Means Algorithm. *Comput. Syst. Sci. Eng.* **2020**, *35*. [CrossRef]
16. Guo, Y.; Cui, Z.; Yang, Z.; Wu, X.; Madani, S. Non-local DWI Image Super-resolution with Joint Information Based on GPU Implementation. *Comput. Mater. Contin.* **2019**, *61*, 1205–1215. [CrossRef]
17. Al-Tawil, K.; Moritz, C.A. Performance Modeling and Evaluation of MPI. *J. Parallel Distrib. Comput.* **2001**, *61*, 202–223. [CrossRef]
18. Lambeck, K. Methods and geophysical applications of satellite geodesy. *Rep. Prog. Phys.* **1979**, *42*, 547–628. doi:10.1088/0034-4885/42/4/001. [CrossRef]
19. Zheng, W.; Zhang, X.; Shu, F. CVN Harddisk System and Software Correlator in e-VLBI Experiments. *Prog. Astron.* **2005**, *23*, 272–286.
20. Deller, A.T.; Tingay, S.J.; Bailes, M.; West, C. DiFX: A Software Correlator for Very Long Baseline Interferometry Using Multiprocessor Computing Environments. *Publ. Astron. Soc. Pac.* **2007**, *119*, 318–336. [CrossRef]
21. Liu, Y.; Zhou, Y.; Zhou, Y.; Ma, L.; Wang, B.; Zhang, F. Accelerating SAR Image Registration Using Swarm-Intelligent GPU Parallelization. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2020**, *13*, 5694–5703. [CrossRef]
22. Zhang, F.; Hu, C.; Li, W.; Hu, W.; Li, H.C. Accelerating time-domain SAR raw data simulation for large areas using multi-GPUs. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 3956–3966. [CrossRef]
23. Hobiger, T.; Kimura, M.; Takefuji, H.; Oyama, T.; Koyama, Y.; Kondo, T.; Gotoh, T.; Amagai, J. GPU Based Software Correlators—Perspectives for VLBI2010. In Proceedings of the Sixth International VLBI Service for Geodesy and Astronomy, Hobart, Australia, 7–13 February 2010; pp. 40–44.
24. Broekema, P.C.; Mol, J.D.; Nijboer, R.; Van Amesfoort, A.S.; Brentjens, M.A.; Loose, G.M.; Klijn, W.F.A.; Romein, J.W. Cobalt: A GPU-based correlator and beamformer for LOFAR. *Astron. Comput.* **2018**, *23*, 180–192. [CrossRef]
25. Yin, Q.; Wu, Y.; Zhang, F.; Zhou, Y. GPU-Based Soil Parameter Parallel Inversion for PolSAR Data. *Remote Sens.* **2020**, *12*, 415. [CrossRef]

26. Zhang, F.; Yao, X.; Tang, H.; Yin, Q.; Hu, Y.; Lei, B. Multiple mode SAR raw data simulation and parallel acceleration for Gaofen-3 mission. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 2115–2126. [CrossRef]
27. Zhang, F.; Li, G.; Li, W.; Hu, W.; Hu, Y. Accelerating spaceborne SAR imaging using multiple CPU/GPU deep collaborative computing. *Sensors* **2016**, *16*, 494. [CrossRef]
28. Li, Z.; Su, D.; Zhu, H.; Li, W.; Zhang, F.; Li, R. A Fast Synthetic Aperture Radar Raw Data Simulation Using Cloud Computing. *Sensors* **2017**, *17*, 113. [CrossRef] [PubMed]
29. Wang, Z.; Hou, Z.; Zhang, Y. Improvement of the Long-Term Orbit Prediction for LEO Navigation Satellites Using the Inner Formation Method. *IEEE Trans. Aerosp. Electron. Syst.* **2019**, *55*, 2532–2542. [CrossRef]
30. Li, Y.-H.; Gao, Y.-L. Satellite orbit prediction based on two-stage filter. In Proceedings of the 2011 3rd International Conference on Advanced Computer Control, Harbin, China, 18–20 January 2011; pp. 44–47. [CrossRef]
31. Khan, D.; Ali, G.; Khan, A.; Chu, Y.; Nisar, K. A New Idea of Fractal-fractional Derivative with Power Law Kernel for Free Convection Heat Transfer in a Channel Flow between Two Static Upright Parallel Plates. *Comput. Mater. Contin.* **2020**, *65*, 1237–1251. [CrossRef]
32. DeviRangaLakshmi, A.; Inabithini, S.R.; Venkataramana, P. Realization of signal processing algorithms using Intel integrated performance primitives (IPP). In Proceedings of the 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), Coimbatore, India, 17–18 March 2017; pp. 1–4.
33. Swaroop, K.V.S.; Rao, K.R. Performance analysis and comparison of JM 15.1 and Intel IPP H.264 encoder and decoder. In Proceedings of the 2010 42nd Southeastern Symposium on System Theory (SSST), Tyler, TX, USA, 7–9 March 2010; pp. 371–375.